関係データベースを利用した並列化コンパイラの開発

2D−7

金丸 弘樹　宇野 総一　森田 和徳　岩根 雅彦
九州工業大学 工学部

1. はじめに

コンパイラには、最適化などの複雑な処理過程が存在する。このため、処理するためのデータ構造も複雑になる事が多い。関係データベース（以下、ＲＤＢ）を用いることでこれらを緩和できる。構成例として、スーパースカラ・プロセッサをターゲットとしたＣ言語並列化コンパイラの開発を行っている。

2. 基本概念

2.1. 木構造とデータベース

構造化言語であるC言語は、コンパイル中のデータの保持形式（データ構造）が主に木構造になる。RDB上でこれを実現するには、木構造のノードにあたる要素と依存関係を示す要素を別のテーブル上に表現する。単純な例を挙げると、図 1のようになる。親・子に限らずノード自身の情報とそれに付随する情報(Opt)はノード情報テーブルに格納される。ノード間の依存は、ノード情報テーブルのIDを利用して依存情報テーブルに保存される。もし、ノード間の依存に特別な関係がある場合は、依存情報テーブルを拡張し、その関係を示す情報を保存する事もできる。子ノード間の関係は、ノード情報テーブルに Order として保存する。このようにして、ツリーで表現される部分（式間の関係・ステートメント間の関係など）は、RDB上に実現できる。

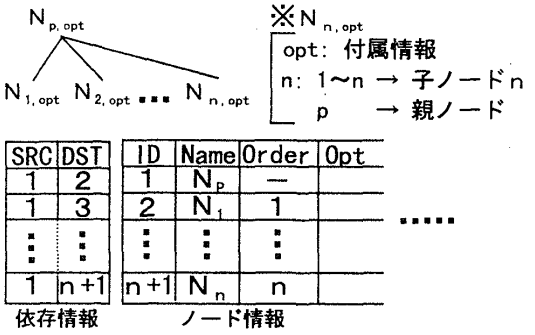

図 1 木構造と関係データベース

2.2. コンパイラ基本構成

図 2 にコンパイラの基本構成を示す。データベースを利用した最適化処理等は原始プログラムの前処理系として動作する。前処理系の出力する中間プログラムは、図3に示されるように、原始プログラムの一部をC言語のasm記述に置き換えたものになる。中間プログラムを処理するバックエンドコンパイラは、asmの記述をサポートしたGNU CCを用いている。GNU CCは、C言語で記述された式(変数)をオペランドに持つアセンブラ命令をasmによってサポートするので図のような置き換えが可能になる。

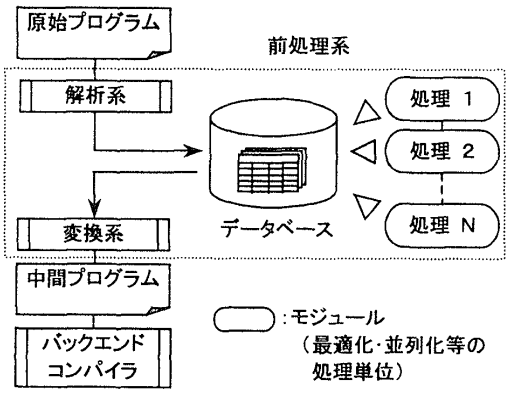

図 2 コンパイラの基本構成

```
void Exam {                    void Exam {
  int a = 1,b = 2;               int a = 1,b = 2;
  int c = 3,d = 4;               int c = 3,d = 4;

  c = a + b;                     asm("ld %0,r0"::"m"(a));
  d = c * b;                     asm("ld %0,r1"::"m"(b));
                                 asm("add r0,r1,r0");
                                 asm("st r0,%0"::"m"(c));
                                 asm("mul r0,r1,r0");
                                 asm("st r0,%0"::"m"(d));

  return d;                      return d;
}                              }
 <原始プログラム>               <中間プログラム>
```

図 3 前処理系による原始プログラム変換例

基本的な動作は、まず、原始プログラムを解析系に通してテーブル表現に置きかえ、データベースへ格納する。いくつかのモジュールによる処理が完了後、変換系を通して中間プログラムを生成する。最後に、中間プログラムをバックエンドコンパイラへ渡すことにより、目的プログラムへ変換される。

3. 並列化プリプロセッサ

3.1. 概略

図 2の基本構成にしたがって、前処理系にあたる並列化プリプロセッサを設計した。今回設計したプリプロセッサ

Development of the parallelized compiler using the relational database
Hiroki Kanamaru,Souichi Uno,Kazunori Morita,Masahiko Iwane
Department of Electrical Engineering,Faculty of Engineering,Kyusyu Institute of Technology

は並列化アルゴリズムにESH[2]を実装している。図 4にその処理過程を示す。

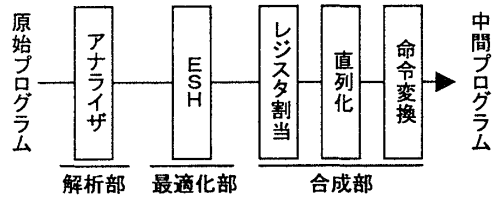

図 4 ＥＳＨによる並列化コンパイラの処理過程

3.2. アナライザ

アナライザは図 2中の解析系にあたり、原始プログラムに対して、字句解析・構文解析・意味解析の処理を行い、これをテーブル表現で出力する。実装には、lex と yacc を利用している。これらの処理は、言語仕様が変更されない限り、その処理はほとんど変化しない。したがって、アナライザは、外部の実行プログラムとして実装した。

3.3. スケジューリング

ESHは、同一優先順位を持つ多項式を変数の定義時間の早いものから二項演算として抽出してスケジューリングを行う。この過程において、式の依存解析、同一優先順位の演算をもつ多項式抽出、演算優先度の計算、2 分木表現からタプル(変数とその前に置かれる演算子の組)表現への変換などの処理が行われる。これらは、前処理系において、各々1つのモジュールとして実装している。

3.4. コード生成

開発中のコンパイラは、スーパースカラ・プロセッサをターゲットとしているため、コード生成時には、並列スケジューリングによって並列化した命令列を直列化する必要がある。また、命令列に対して対象マシン別にレジスタ割り当てなどの作業が必要になる。これらの処理をコード生成時に行う。この後、スケジューリングされた原始プログラムは、対象マシンのアセンブラに直され、C言語の asm 記述によって中間プログラムとして出力される。

4. 開発および実行環境

今回、前処理を行うためのRDBアプリケーションとして、Access97を利用した。したがって、前処理系にあるモジュールは、Visual BASIC(以下、VB)、および、SQLを用いて記述されている。

SQLを利用することで、モジュールに必要なプログラム量を減らすことができる。これは、モジュール内の前処理段階でSQLを実行し、そのモジュールに必要な情報を必要な形で先に取得できれば、同じ情報を集めるためのアクセス手順の記述が不要になるからである。また、SQLに不得手な分岐・反復などの処理は、VBが支援するので、複雑な処理にも柔軟な対応が可能になる。

並列コンパイラの実行は、GUIの操作パネルによって行われる。操作パネルからユーザーの指定によって各モジュールを起動し、データベースに対して処理させる。モジュールによる処理の結果は、操作パネルのウィンドウに反映されるように実装した。

図 5に、各モジュールの実行結果を示す。これは、各モジュールの実現に要したプログラム量とそれに式(Ｉ)の例を処理させた時のＳＱＬ実行回数である。なお、表に上げるモジュールについては、データベースの更新をＳＱＬのみで行った。

処理内容	VB	SQL
同一優先順位の演算を持つ多項式の抽出	61	22
演算優先度の計算	64	20
タプル表現への変換	50	27
式の依存解析	36	11

SQL：SQL の実行回数
VB：処理プログラムの量（行数）

$$\begin{cases} X=(A\div B\times C+D\div(-E))-F+(-G) \\ Y=X\times A+B+C \\ Z=X\times Y \end{cases} \quad \cdots (\mathrm{I})$$

図 5 プログラム量とＳＱＬ実行回数

ＳＱＬ文の実行は、非常に計算コストが高いため、通常のコンパイラに比べコンパイル速度にかなりの遅延が生じる。しかし、各モジュールの処理は、図 5に見られる程度の記述で実現できている。特に、処理の対象が式単位である場合には、テーブル上の式に関する情報に絞って処理を行うことができ、それ以外の制御構造など部分について知る必要がなくなる。また、各モジュールの依存度を低くプログラムできれば、モジュールによる処理順序の入れ替えなども可能になる。

5. むすび

RDBを利用した並列化プリプロセッサの開発について述べた。今後は、スペックの拡張などについて検討していきたい。

謝 辞

研究を進めるにあたり、御助力頂いた（株）東芝日野工場に深謝いたします。

参 考 文 献

[1] 佐々 政孝,「岩波講座 ソフトウェア科学 5 プログラミング言語処理系」, 岩波書店, １９９４

[2] 岩根他,「式の分割による並列化アルゴリズム ESH とその評価」, 情報処理学会論文誌 vol. 38, No. 9